# 第3章 程序设计初步

3.1 堆栈的作用

3.2 算术逻辑运算指令

**3.3 分支程序设计**

3.4 循环程序设计

3.5 子程序设计

# 3.3 分支程序设计

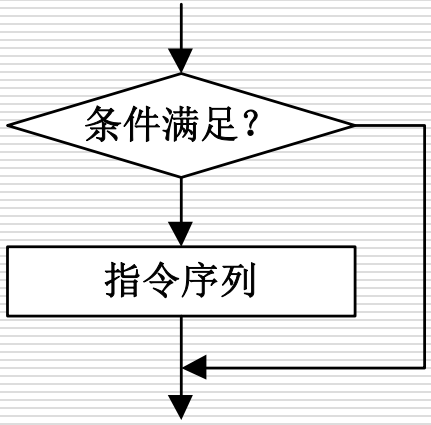
## 3.3.1 分支程序设计示例

## 3.3.2 无条件和条件转移指令

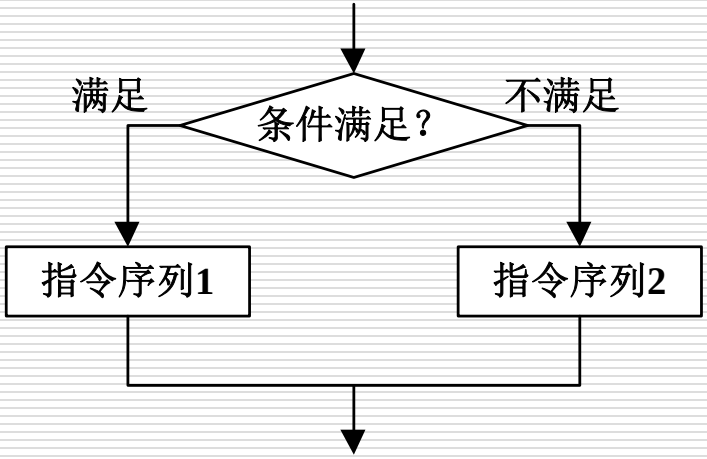
## 3.3.3 多路分支的实现

# 3.3.1 分支程序设计示例

- 两种分支结构




(a)




(b)

# 3.3.1 分支程序设计示例

## ➢演示函数cf315

字符小写化：
如为大写字母，则转成小写字母

```
int  cf315(int  ch)
{
    if ( ch >= 'A' && ch <= 'Z' )
        ch += 0x20;
    return  ch;
}
```

整型

大小写字母的ASCII码
相差20H

# 3.3.1 分支程序设计示例

➢演示函数cf315

不采用编译优化，
目标代码

```
    push   ebp
    mov    ebp, esp                    ;建立堆栈框架
                                       ; if ( ch>='A' && ch<='Z' )
    cmp    DWORD PTR [ebp+8], 65
    jl     SHORT  LN1cf315             ;小于，则跳转
    cmp    DWORD PTR [ebp+8], 90
    jg     SHORT  LN1cf315             ;大于，则跳转
                                       ; ch += 0x20;
    mov    eax, DWORD PTR [ebp+8]
    add    eax, 32
    mov    DWORD PTR [ebp+8], eax
LN1cf315:                              ; return  ch;
    mov    eax, DWORD PTR [ebp+8]      ;返回值
    pop    ebp                         ;撤销堆栈框架
    ret                                ;返回
```

**[ebp+8]**
参数**ch**（整型）

# 3.3.1 分支程序设计示例

➢演示函数cf315

编译优化：速度最大化
目标代码

```
    push   ebp
    mov    ebp, esp                    ;建立堆栈框架
                                       ;if ( ch>='A' && ch<='Z' )
    mov    eax, DWORD PTR [ebp+8]
    lea    ecx, DWORD PTR [eax-65]
    cmp    ecx, 25
    ja     SHORT  LN1cf315

    add    eax, 32
LN1cf315:                              ;return  ch;
    pop    ebp                         ;撤销堆栈框架
    ret
```

[ebp+8]
参数ch

两次比较
转化为一次比较

大写转小写

# 3.3.1 分支程序设计示例

➢演示函数cf316

把一位十六进制数转换为对应ASCII码

$$y = \begin{cases} x + 30H & (0 <= x <= 9) \\ x + 37H & (0AH <= x \end{cases}$$

```
int  cf316(int  m)
{
    m = m & 0x0f;      //确保
    if ( m < 10 )
        m += 0x30;     //数字
    else
        m += 0x37;     //字母A
    return  m;
}
```

整型

分支

| | | |
|---|---|---|
| 30 | 00110000 | 0 |
| 31 | 00110001 | 1 |
| 32 | 00110010 | 2 |
| 33 | 00110011 | 3 |
| 34 | 00110100 | 4 |
| 35 | 00110101 | 5 |
| 36 | 00110110 | 6 |
| 37 | 00110111 | 7 |
| 38 | 00111000 | 8 |
| 39 | 00111001 | 9 |
| 3A | 00111010 | : |
| 3B | 00111011 | ; |
| 3C | 00111100 | < |
| 3D | 00111101 | = |
| 3E | 00111110 | > |
| 3F | 00111111 | ? |
| 40 | 01000000 | @ |
| 41 | 01000001 | A |

# 3.3.1 分支程序设计示例

➢演示函数cf316

不采用编译优化

```
    push   ebp
    mov    ebp, esp                          ;建立堆栈框架
                                             ;m = m & 0x0f;
    mov    eax, DWORD PTR [ebp+8]
    and    eax, 15
    mov    DWORD PTR [ebp+8], eax
                                             ;if ( m < 10 )
    cmp    DWORD PTR [ebp+8], 10
    jge    SHORT LN2cf316

                                             ;m += 0x30;
    mov    ecx, DWORD PTR [ebp+8]
    add    ecx, 48
    mov    DWORD PTR [ebp+8], ecx
    jmp    SHORT  LN1cf316
LN2cf316:
```

[ebp+8]
参数m

两路分支的合并

# 3.3.1 分支程序设计示例

➢演示函数cf316

不采用编译优化

```
LN2cf316:
                                        ;m += 0x37;
    mov    edx, DWORD PTR [ebp+8]
    add    edx, 55
    mov    DWORD PTR [ebp+8], edx
LN1cf316:
                                        ;return  m;
    mov    eax, DWORD PTR [ebp+8]       ;EAX含返回值
    pop    ebp                          ;撤销堆栈框架
    ret
```

[ebp+8]
参数m

两路分支合并点

# 3.3.1 分支程序设计示例

➢演示函数cf316

编译优化：
速度最大化

```
    push   ebp
    mov    ebp, esp                        ;建立堆栈框架
    mov    eax, DWORD PTR [ebp+8]          ;m = m & 0x0f;
    and    eax, 15
    cmp    eax, 10                         ;if ( m < 10 )
    jge    SHORT LN2cf316
    add    eax, 48                         ;m += 0x30;
    pop    ebp
    ret
LN2cf316:
    add    eax, 55                         ; m += 0x37;
    pop    ebp
    ret
```

[ebp+8]
参数m

2条返回指令
这样处理的优点？

# 3.3.1 分支程序设计示例

## ➢演示函数cf317

观察优化C程序后的结果

```
int  cf317(int  m)
{
    m = m & 0x0f;
    m +=  0x30;
    if ( m > '9' )
        m +=  7;
    return  m;
}
```

分支结构：单路

# 3.3.1 分支程序设计示例

➢演示函数cf317

编译优化：
速度最大化

```
    push   ebp
    mov    ebp, esp
    mov    eax, DWORD PTR [ebp+8]
    and    eax, 15                  ;m = m & 0x0f;
    add    eax, 48                  ;m +=  0x30;
    cmp    eax, 57                  ;if ( m > '9' )
    jle    SHORT  LN1cf317
    add    eax, 7                   ;m +=  7;
LN1cf317:
    pop    ebp
    ret
```

只有一个条件转移指令

# 3.3.2 无条件和条件转移指令

## ➢相关基本概念

按转移是否跨段来区分

✓ 段内转移和段间转移：

- **段内转移**是仅仅重新设置指令指针寄存器**EIP**的转移。由于没有重置代码段寄存器**CS**，所以转移后继续执行的指令仍在同一个代码段中。
- **段间转移**是不仅重新设置**EIP**，而且重新设置代码段寄存器**CS**的转移。由于重置**CS**，所以转移后继续执行的指令在另一个代码段中。
- 段内转移也被称为**近转移**，段间转移也被称为**远转移**。

# 3.3.2 无条件和条件转移指令

## ➢相关基本概念

✓ 转移的约束：

- 条件转移指令和循环指令，只能实现段内转移。
- 无条件转移指令和过程调用及返回指令，既可以是段内转移，也可以是段间转移。
- 软中断指令和中断返回指令一定是段间转移。

循环指令（下一节讲解）
软中断指令和中断返回指令（今后介绍）

# 3.3.2 无条件和条件转移指令

## ➢相关基本概念

给出转移目的地址的方式

✓ 直接转移和间接转移：

- 在转移指令中直接给出转移目的地址的转移被称为**直接转移**。
- 在转移指令中没有直接给出转移目的地址，但给出包含转移目的地址的寄存器或者存储单元，这样的转移被称为**间接转移**。

无条件转移指令
过程调用指令

# 3.3.2 无条件和条件转移指令

## ➢无条件转移指令

跨段（否/是）
目的地址（直接/间接）

✓ 无条件转移指令可分为**4**种：

- 段内直接转移
- 段内间接转移
- 段间直接转移
- 段间间接转移

✓ 无条件转移指令均不影响标志寄存器中的状态标志

# 3.3.2 无条件和条件转移指令

## ➢无条件转移指令（JMP）

✓ 无条件段内直接转移指令的一般格式

```
JMP    LAB
```

标号LAB用于表示转移的目标位置，或者说转移目的地。

在2.6.4节介绍过
无条件转移指令

# 3.3.2 无条件和条件转移指令

## ➢无条件转移指令（JMP）

✓ 无条件段内直接转移指令的机器码格式

| 操作码OP | 地址差rel |
|---|---|

- 地址差rel，是转移目标地址偏移（标号LAB所指定指令的地址偏移）与紧随JMP指令的下一条指令的地址偏移之间的差值。
- 执行无条件段内转移指令时，把指令中的地址差rel加到指令指针寄存器EIP上，使EIP之内容为转移目标地址偏移，从而实现转移。

# 3.3.2 无条件和条件转移指令

## ➢无条件转移指令（JMP）

✓ 无条件段内直接转移指令的机器码格式

| 操作码OP | 地址差rel |
| --- | --- |

- 地址差rel，可用1字节表示，也可用4字节（或2字节）表示。如只用1字节表示，就称之为**短（short）转移**；否则就称之为**近（near）转移**。
- 由于差值是有符号数，所以可以实现向前方（未来）转移，也可以实现向后方（过往）转移。

# 3.3.2 无条件和条件转移指令

## ➢无条件转移指令（JMP）

✓ 无条件段内直接转移指令的机器码格式

在VC2010嵌入汇编环境中，保护方式，32位代码段。

- 一字节表示的地址差的范围比较有限。
- 在保护方式下（32位代码段），地址差也可以用32位来表示，这样就可以转移到段内的任何有效目标地址。
- 当汇编器能够正确地计算出地址差rel，则根据地址差的大小，决定采用1字节表示，还是采用4字节（或2字节）表示。否则，汇编器可能会用较多的字节数来表示地址差。
- 如在写程序时能估计出用1字节就可表示地址差，那么可在标号前加一个汇编器操作符“SHORT”。

# 3.3.2 无条件和条件转移指令

## ➢无条件转移指令（JMP）

✓ 相对转移

这种利用目标地址与转移指令所处地址之间的差值来表示转移目标地址的转移方式，也被称为**相对转移**。

相对转移有利于程序的浮动！

# 3.3.2 无条件和条件转移指令

## ➢无条件转移指令（JMP）

✓ 无条件段内间接转移指令的一般格式

```
JMP    ORRD
```

- 指令使控制无条件地转移到由操作数OPRD的内容给定的目标地址处。
- 在保护方式下（32位代码段），OPRD是32位通用寄存器或者双字存储单元，其内容直接被装入指令指针寄存器EIP，从而实现转移。

在VC2010嵌入汇编环境中，
保护方式，32位代码段。

# 3.3.2 无条件和条件转移指令

## ➢无条件转移指令（JMP）

✓ 无条件段内间接转移指令的示例

```
JMP    ECX                          ;目标地址是ECX寄存器的内容
JMP    DWORD  PTR  [EBX]            ;目标地址是由EBX所指向双字存储单元内容
```

# 3.3.2 无条件和条件转移指令

演示无条件段内转移指令的使用

➢演示程序dp318

```
#include <stdio.h>
char  flag1='0', flag2='0', flag3='0';
int   ptonext;                    //存放转移地址
int  main( )
{
    _asm  {
        LEA    EAX, STEP2     //取得第二步的开始地址
        MOV    ptonext, EAX   //保存到存储单元
        ;
        LEA    EDX, STEP1     //取得第一步的开始地址
        JMP    EDX            //转移到第一步
        ;
    STEP2:
        MOV    flag2, 'B'
        JMP    STEP3          //转移到第三步
```

段内间接转移

段内直接转移

# 3.3.2 无条件和条件转移指令

➢演示程序**dp318**

```
    ;
    STEP1:
        MOV   flag1, 'A'
        JMP   ptonext                    //转移到第二步
            ;
    STEP3:
        MOV   flag3, 'C'
    }
    printf("%c,%c,%c\n",flag1,flag2,flag3);    //显示为A,B,C
    return  0;
}
```

段内间接转移

根据显示结果，可知执行顺序。

# 3.3.2 无条件和条件转移指令

## ➢无条件转移指令（JMP）

✓ 无条件段间转移指令

- 无条件段间直接转移指令的使用格式与上述无条件段内直接转移指令相类似
- 无条件段间间接转移指令的使用格式和上述无条件段内间接转移指令相类似。
- 由于涉及到改变代码段寄存器CS的内容，所以较为复杂。

以后介绍

# 3.3.2 无条件和条件转移指令

## ➢条件转移指令（Jcc）

✓Jcc指令的一般格式

```
Jcc    LAB
```

符号cc表示各种条件缩写，LAB代表源程序中的标号。

当条件满足时，转移到标号LAB处；否则继续顺序执行。

在2.6.2节介绍过条件转移指令，
列出了所有条件转移指令。

# 3.3.2 无条件和条件转移指令

## ➢条件转移指令（Jcc）

✓ 条件转移指令的进一步说明

- 虽然通常根据标志寄存器中的状态标志判断条件是否满足，但条件转移指令本身的执行不影响状态标志。
- 条件转移指令只局限于段内转移。
- 条件转移指令也采用相对转移方式。

在2.6.2节介绍过条件转移指令，
列出了所有条件转移指令。

# 3.3.2 无条件和条件转移指令

## ➢条件转移指令（Jcc）

✓ 条件转移指令的进一步说明

- 机器码格式类似无条件转移指令。

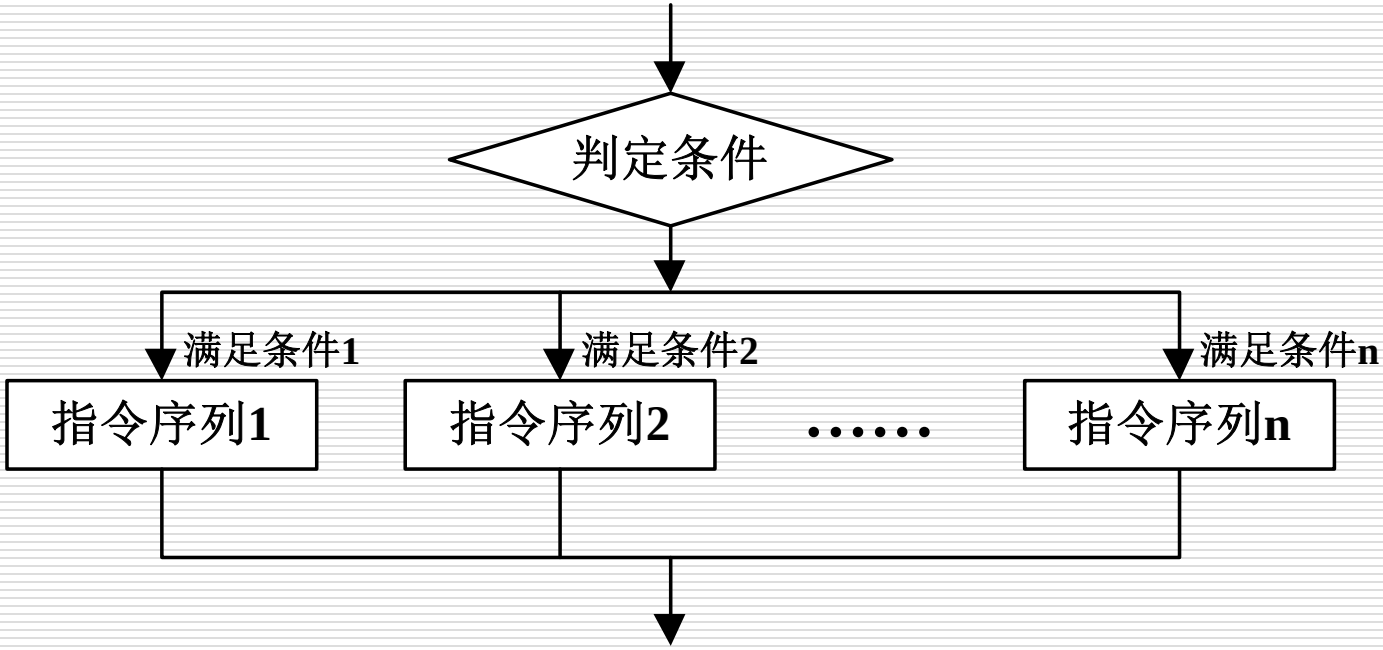
| 操作码OP | 地址差rel |
|---|---|

- 当条件满足时，把这个差值加到EIP上，从而使EIP等于标号所代表的偏移，从而实现转移。
- 由于差值是一个有符号数，所以条件转移指令可以实现向前方（未来）转移，也可以实现向后方（过往）转移。
- 地址差rel可用1个字节表示，也可用4个字节（或2个字节）表示。

类似于 无条件转移指令

# 3.3.3 多路分支的实现

- 多路分支结构



# 3.3.3 多路分支的实现

➢演示函数cf319

演示多路分支

```
int  cf319(int x,  int operation)
{
    int  y;
    //多路分支
    switch ( operation ) {
        case 1:
            y = 3*x;
            break;
        case 2:
            y = 5*x+6;
            break;
        case 4:
        case 5:
            y = x*x ;
            break;
        case 8:
            y = x*x+4*x;
            break;
        default:
            y = x ;
    }
    if ( y > 1000 )
        y = 1000;
    return  y;
}
```

为了演示故意缺少
case 3、6和7等情形

# 3.3.3 多路分支的实现

➢演示函数cf319

编译优化：
速度最大化

```
push   ebp
mov    ebp, esp
                                          ; switch ( operation ) {
mov    eax, DWORD PTR [ebp+12]   ;取得参数operation（case值）
dec    eax                       ;从0开始计算，所以先减去1
cmp    eax, 7                    ;从0开始计算，最多就是7
ja     SHORT LN2cf319            ;超过，则转default
;
jmp    DWORD  PTR  LN12cf319[ eax*4 ]      ;实施多路分支
;
```

段内间接 无条件转移
LN12cf319地址表（每项4字节）

[ eax*4+ LN12cf319 ]

# 3.3.3 多路分支的实现

## ➢演示函数cf319

```
LN12cf319:                          ;多向分支目标地址表
    DD    LN6cf319                  ; case 1
    DD    LN5cf319                  ; case 2
    DD    LN2cf319                  ; default
    DD    LN4cf319                  ; case 4
    DD    LN4cf319                  ; case 5
    DD    LN2cf319                  ; default
    DD    LN2cf319                  ; default
    DD    LN3cf319                  ; case 8
```

LN12cf319地址表（每项4字节）

# 3.3.3 多路分支的实现

## ➢演示函数cf319

```
LN6cf319:                                   ; case 1:
                                            ; y = 3*x;
    mov    eax, DWORD PTR [ebp+8]
    lea    eax, DWORD PTR [eax+eax*2]
    jmp    SHORT  LN7cf319                  ; break;
    ;
LN5cf319:                                   ; case 2:
                                            ; y = 5*x+6;
    mov    eax, DWORD PTR [ebp+8]
    lea    eax, DWORD PTR [eax+eax*4+6]
    jmp    SHORT  LN7cf319                  ; break;
    .
```

**[ebp+8]**
**参数x**

**Break语句的代码**

情形1和2处理的对应代码

# 3.3.3 多路分支的实现

## ➢演示函数cf319

```
LN4cf319:                               ; case 4:
                                        ; y = x*x ;
    mov    eax, DWORD PTR [ebp+8]
    imul   eax, eax
    jmp    SHORT LN7cf319               ;  break;
    ;
LN3cf319:                               ; case 8:
                                        ; y = x*x+4*x;
    mov    ecx, DWORD PTR [ebp+8]
    lea    eax, DWORD PTR [ecx+4]
    imul   eax, ecx
    jmp    SHORT LN7cf319               ; break;
    ;
```

**[ebp+8]**
**参数x**

情形4和7处理的对应代码

# 3.3.3 多路分支的实现

## ➢演示函数cf319

```
LN2cf319:                                   ; default:
                                            ; y = x ;
    mov    eax, DWORD PTR [ebp+8]
                                            ; }
LN7cf319:                                   ; if ( y > 1000 )
    cmp    eax, 1000
    jle    SHORT  LN1cf319
                                            ; y = 1000;
    mov    eax, 1000
LN1cf319:                                   ; return  y;
    pop    ebp                              ;撤销堆栈框架
    ret
    ;
```

结束返回